Author: MCUXpresso IDE Team

Version: 2.0.1 10th August 2018

Using the MIMXRT1050-EVK(B) with MCUXpresso IDE v10.2.1

Table of Contents

1 Overview

This document is intended to assist users who are new to using the MIMXRT1050-EVK(B) board. It assumes some familiarity with creating and debugging projects with MCUXpresso IDE v10.2.1. It is also assumes an SDK for the MIMXRT1050-EVK(B) board has been installed into MCUXpresso IDE, and the LinkServer/CMSIS-DAP debug connection (DAPLink) will be used for all debug operations.

Note: it is strongly recommended that the latest available SDK is installed – at the time of writing this is the EVKB-IMXRT1050 version 2.4.2.

For more information on using MCUXpresso IDE, please see the MCUXpresso IDE User Guide.

Note: this is a guide only and not intended as a definitive document

2 Read Me First

Some early EVK boards shipped with an image in flash (hyperflash) that when run can prevent a successful debug connection. Therefore, it is strongly recommended that the flash is first erased.

To erase this flash, please follow the procedure described in the Troubleshooting section at the end of this document before any other debug operations are attempted.

Please also refer to the section DAPLink Firmware version to ensure your board is programmed with the correct debug probe firmware.

IMXRT1050 EVK and EVKB

The original IMXRT1050 EVK board has been replaced by an updated EVK B variant. This new board contains a later revision of the IMXRT1050 silicon and also various improvements including observed improved debug stability when using the external 20 way JTAG header.

A new SDK – MIMXRT1050-EVKB, is available to support this later board. It is recommended that this SDK is used in preference to earlier SDKs whether debugging EVK or EVK(B) boards.

For further information on the I.MX-RT1050 and the EVK development boards please see the following link: https://www.nxp.com/pages/:i.MX-RT1050

For guidance migrating from Rev A0 to Rev A1 silicon, please see: https://www.nxp.com/docs/en/nxp/application-notes/AN12146.pdf

SDK Changes

SDK_2.x_EVKB-IMXRT1050 version 2.4.x incorporates a number of changes and improvements from previous SDK versions. Some of the significant changes from the point of view of general build and debug are listed below:

- Example projects now typically target execution from external flash
  - previously most examples linked to execute from RAM
- Onchip SRAM is typically chosen for data storage rather than SDRAM
- Example projects now default to output text via the SDK Debug Console UART
  - this text output can be captured using the DAPLink built in VCOM support (not available when using an external debug probe)
    - MCUXpresso IDE now includes a terminal view suitable for capturing this output within the IDE. Please see the MCUXpresso IDE User Guide section 18.9 'Using Terminal View for UART communication with target'
  - this default can of course be changed during example import
- MPU memory configurations regions settings are improved

Debug Restrictions

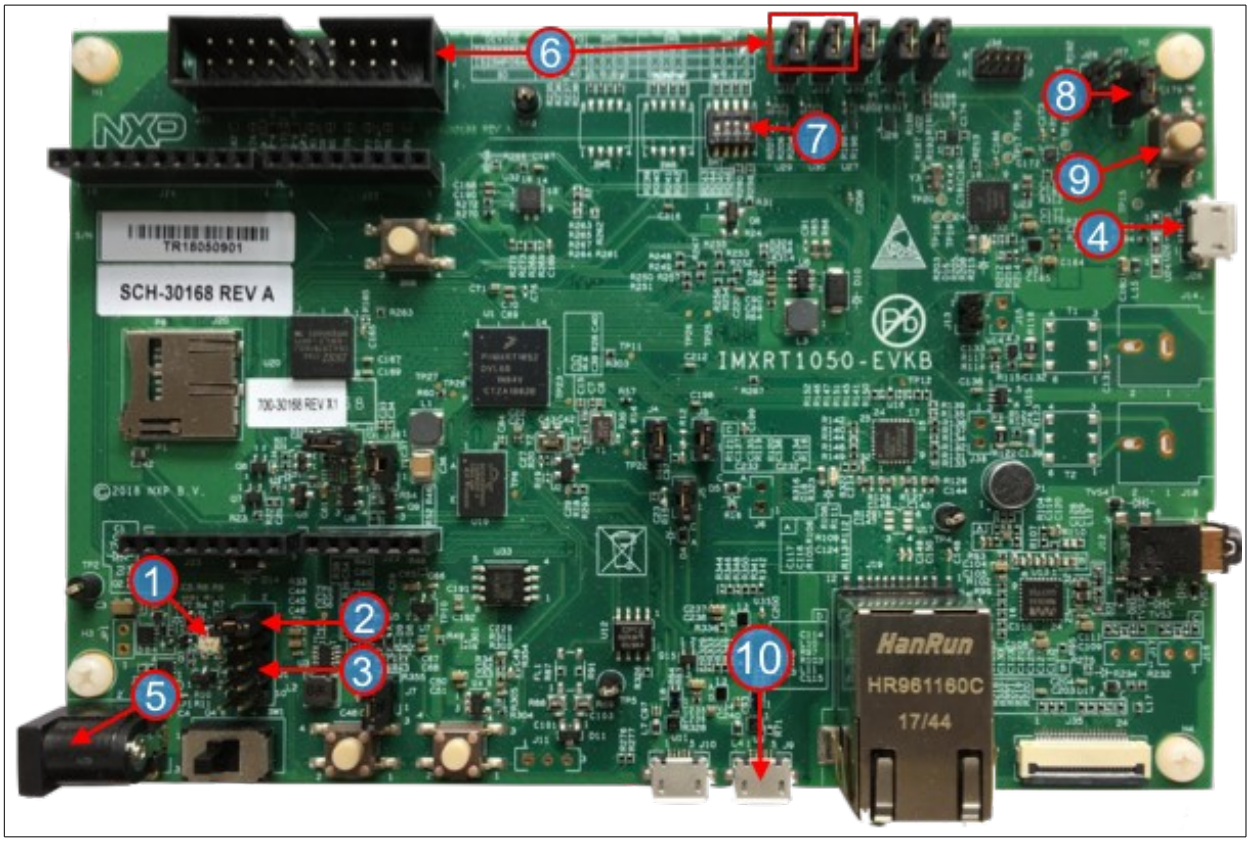
Despite the extensive feature set of this MCU/Board, some debug capabilities are not available. While the MCU does support SWO, due to multiplexing issues this feature is not available on the EVK(B) boards. Also ETB based instruction trace is not supported by this MCU.

Also supplied with this Document

- LinkServer Connect Script to initialise SD-RAM
- LinkServer Connect Script to recover debug connections if the default connection mechanism fails
- QSPI XIP header files for EVK(B) board modified to use QSPI flash

3 Overview of Board features related to Debug

Below is a photo of the MIMXRT1050-EVKB board with key elements numbered and described:



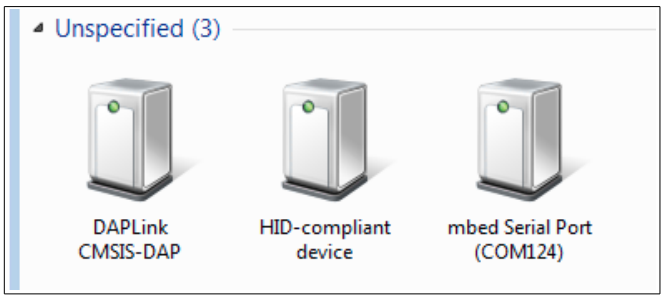
1. Power LED - **If this LED is not lit, the MCU/board is not powered and cannot be debugged.**
2. J1 - Link position (top) for board powered externally (see 5).
3. J1 - Link position (middle) for board powered via the OpenSDA DAPLink debug probe connection (see 4).
4. J28 - OpenSDA DAPLink (CMSIS-DAP) connection (see 3). This is the recommended debug connection for initial use.
5. J2 - External power connection. If used, this should be 5v, centre +ve, and J1 set (see 2). This allows power to the board to be controlled via switch SW1 (to the right of (5)).
6. 20 way JTAG style connection for use with external debug probes such as the LPC-Link2. External debug probes will typically achieve significantly faster debug operations than the onboard OpenSDA debug probe. *Note: highlighted links will disable OpenSDA debug if removed. This may improve external debug probe stability.*
7. SW7 - DIP switches to select boot options. Initially this should be set for boot from Hyperflash: so set as 1-off, 2-on, 3-on, 4-off.
8. J27 – OpenSDA Bootloader selection. Set jumper 1-2 for bootloader mode, 2-3 for OpenSDA debug.
9. SW4 – Reset for use with OpenSDA firmware programming when J27 is set 1-2.
10. Target USB – can also be used to provide power to the target if J1 – Link position set between (top and middle).

4 Debug Connection

For initial use, we recommend using the OpenSDA USB connection (see 4 above) for the first debug operation(s).

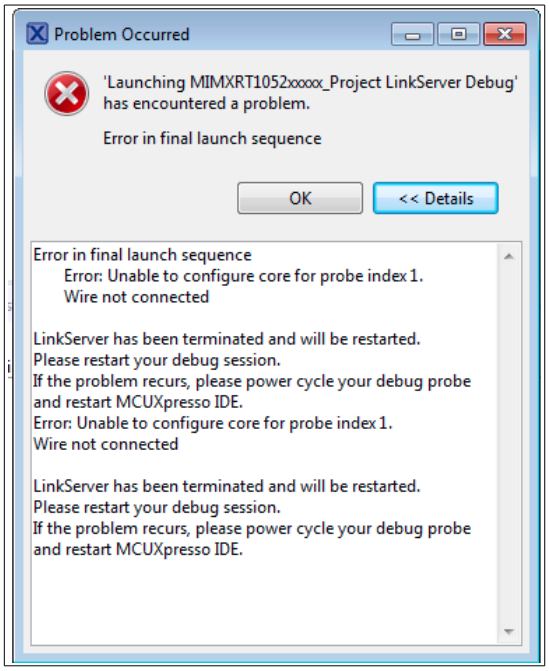
Boards are delivered with DAPLink debug probe firmware (CMSIS-DAP) pre-programmed into the OpenSDA hardware (please also see DAPLink Firmware version below).

Once an OpenSDA USB connection has been made (on Windows), 'Devices and Printers' will show the devices as below:



This connection can also power the board if the J1 link is set to the mid position (see point 2 in Overview of Board features ... above).

Note: if this link is set correctly, the LED next to J1 will light green. If the LED is not lit, the MCU will not be powered and debug will fail, resulting in an error similar to that below (despite the DAPLink debug connection being available):



5 DAPLink Firmware version

Some of the initial shipments of boards contain a version of DAPLink firmware which has known issues. To check the version on your board, simply make a USB connection to the OpenSDA USB connection (as described above) and open a filer window on the EVK(B)-MIMXRT drive.

Open the file DETAILS.TXT on this drive and confirm the 'Interface Version:' is 0244 or greater.

If this is not the case, then the firmware should be updated following the procedure detailed below.

Updating the DAPLink firmware

From MCUXpresso IDE go to Help -> Additional resources -> OpenSDA Firmware Updates, this will open a web page 'OPENSDA: OpenSDA Serial and Debug Adapter'. From this page, locate the dropdown and select the MIMXRT 1050-EVK board.

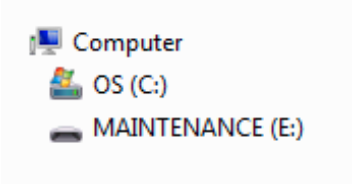
Note: at the time of writing only EVK boards are listed here.

Download the latest DAPLink binary – at the time this document was created this reports as DAPLink v0244.

To install the firmware, follow the procedure below:

1. Power off the board
2. Locate J27- identified in the board features as (8).
    a. Set the J27 Jumper between pins 1 and 2 (this is the lower position with the board oriented as the photograph above).
3. Press and hold SW4 – identified in the board features as (9).
4. Connect a USB cable to the OpenSDA connector
5. Release SW4

6. Open a filer window and observe a drive labelled MAINTENANCE appears:



7. Open this drive and drag the previously downloaded firmware onto the filer window
    a. The filer window should close when the firmware update has completed
8. Eject the device
9. Power off the board
10. Restore the J27 Jumper to between pins 2-3 (this is the upper position with the board oriented as the photograph above).

Now confirm the updated version number as described at the beginning of this section.

Important Note: *Two versions of OpenSDA firmware are available, one for onboard HyperFlash (fitted as standard to the MIMXRT1050-EVK(B) boards) and one for onboard QSPI-flash (only accessible to reworked boards). These version indicate the flash type that will be supported for the boards when programmed via drag and drop - mass storage devices.* ***For users of standard boards be sure to install the HyperFlash version.***

6 Memories

Below is a list of the usable memories on this MCU and board. Some, or all of these regions will be visible within an MCUXpresso IDE project's Memory Configuration Editor (as below):

Type	Name	Alias	Location	Size	Driver
Flash	BOARD_FLASH	Flash	0x60000000	0x4000000	MIMXRT1050-EVK_S26KS512S.cfx
RAM	SRAM_DTC	RAM	0x20000000	0x20000	
RAM	SRAM_ITC	RAM2	0x0	0x20000	
RAM	SRAM_OC	RAM3	0x20200000	0x40000	
RAM	BOARD_SDRAM	RAM4	0x80000000	0x2000000	

- *External board memories are highlighted in blue. LinkServer Flashdriver for the hyperflash device highlighted in red.*

- *Note: By default, projects will be linked to the first flash memory in this list and use the first RAM region for data, heap and stack. However, the SDK (projects and examples) may select a subset of these memories and/or change their order to control linkage (and also override default linkage using the LinktoRAM feature – see later).*

Board_Flash (Hyperflash) at 0x60000000: This 64MB device is board memory external to the MCU. Programming of this device is provided by a flash driver called MIMXRT1050-EVK_S26KS512.cfx (for LinkServer CMSIS-DAP debug connections). On reset, (if SW7 is set for Flash Boot) the BootROM will interrogate this device and attempt to identify a specific image header, if found, the header data will be used to configure its operation (and also initialise the SDRAM). If a correct header is not found, this device will be unavailable.

Note: MCUXpresso IDE will automatically generate and locate an appropriate header from information supplied by the SDK for new projects and as required for example projects.

Code can be run directly from this Flash, this is known as Execute in Place (XIP). This Flash can be cached by the MCU.

Note: The term XIP is used to differentiate from an alternative boot strategy, where the BootROM will relocate code (and data) from flash for RAM execution. This mode of operation is not directly supported within MCUXpresso IDE v10.2.x.

Note: Also see the section on Flash Drivers.

SRAM_OC at 0x20200000. This 256KB (FlexRAM) is on chip SRAM accessed over AXI and is cacheable by the MPU. Code or data accessed from this memory will use space within the cache. If this memory is marked as not cacheable, performance will be significantly reduced.

SRAM_ITC at 0x0: This 128KB device (FlexRAM) is on chip SRAM, and tightly coupled to the MCU and will 'seen' by the CPU before the cache, therefore the contents of this RAM will not cached. This RAM will provide the best deterministic performance for program execution.

SRAM_DTC at 0x20000000.: This 128KB (FlexRAM) is on chip SRAM, and tightly coupled to the MCU and will 'seen' by the CPU before the cache, therefore the contents of this RAM will not be cached. This RAM will provide the best deterministic performance for data accesses.

Note: Tightly coupled memories may be described to the MPU with cacheable attributes, however their contents will not actually be cached. They are intended to be used for code (and data) requiring the maximum deterministic performance (and minimum power - this is a complex area and will not be discussed further in this document).

SDRAM at 0x8000000: This 32MB device is board memory external to the MCU. This RAM block must be initialised before it can be used. An XIP Flash header contains the data for the BootROM to use to initialise this RAM memory. Alternatively, if a project is targeted to run from this RAM, a debug Script can be run to initialise this device.

Note: If this initialisation does not occur, then the RAM will not be available and a debug operation targeting this memory will fail!.

Code can be run directly from this RAM. This RAM can be cached by the MPU.

Memory attributes and Cache(s)

Complex memory systems present considerable flexibility to any system designer and much of this detail is beyond the scope of this document. However, it should be understood that this MCU contains a cache designed to improve the system performance when accessing board memories (SDRAM, Flash and OC_RAM). Furthermore, memory regions can be assigned properties governing how memory access are treated inside that region by the CPU.

A Memory Protection Unit (MPU) is present on this MCU to control these memory region properties.

New and example (board) projects will contain a function *BOARD_ConfigMPU* within the file board.c . This function performs the MPU configuration for the various memory regions including cache setup and the exact behaviour of this function is controlled by a number of defined symbols. **It is strongly recommended that (if used) this function is examined and understood to ensure the memory system is configured as desired.**

Note: In SDK versions 2.4 and 2.4.1, the External Flash is configured as ReadWrite:

```
MPU->RBAR = ARM_MPU_RBAR(2, 0x60000000U);
MPU->RASR = ARM_MPU_RASR(0, ARM_MPU_AP_FULL, 0, 0, 1, 1, 0, ARM_MPU_REGION_SIZE_512MB);
```

This is not ideal - it is suggested that **ARM_MPU_AP_RO** is a more appropriate setting for the flash region. This will be corrected in a future SDK release.

Furthermore, since the MPU controls access to actual memory regions, the selected region sizes should match the actual memories, thereby enabling the MPU to trap accesses outside of real memories.

Note: in SDK version 2.4.x the 32MB SDRAM region is configured so the last 2MB will not be cached:

```
MPU->RBAR = ARM_MPU_RBAR(8, 0x81E00000U);
MPU->RASR = ARM_MPU_RASR(0, ARM_MPU_AP_FULL, 1, 0, 0, 0, 0, ARM_MPU_REGION_SIZE_2MB);
```

If a project is created with the SDRAM as the first RAM region then the stack will automatically be located at the end of this region i.e. within this uncached region of memory. Since uncached SDRAM will see hugely degraded performance, this situation should be avoided! Please refer to the MCUXpresso IDE User Guide Section 16.10 *Modifying heap/stack placement* for details on controlling stack placement, alternatively the MPU settings can of course be changed etc.

7 Flash Drivers

The SDKs EVK(B)-IMXRT1050 ships with a hyperflash driver for CMSIS-DAP debug connections. This driver ***MIMXRT1050-EVK_S26KS512S.cfx*** targets a single hyperflash ‘chip’ as fitted as standard to the MIMXRT1050-EVK(B) board. Also supplied are drivers for specific

QSPI and EcoXiP devices, known as ***MIMXRT1050-EVK_IS25WP064A.cfx*** and ***MIMXRT1050-EcoXiP_ATXP032.cfx*** respectively.

New in MCUXpresso IDE version 10.2.1 is the source project for these driver located at: *<IDE install Directory>\ide\Examples\Flashdrivers\NXP\iMXRT.* These driver project are supplied as a base for users to develop CMSIS-DAP flash drivers for alternative flash devices.

Also new in MCUXpresso IDE version 10.2.1 are two drivers that self configure from flash **JEDEC SFDP** data, and are supplied in binary form only:

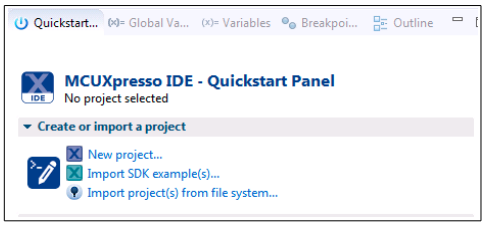
MIMXRT1050_SFDP_HYPERFLASH.cfx
MIMXRT1050_SFDP_QSPI.cfx

These drivers are located at: *<IDE install Directory>\ide\bin\Flash* and should be used in preference for any new project targeting flash devices that supports the JEDEC SFDP standard. Please see the MCUXpresso IDE User Guide Section 14.2.4 for more information.

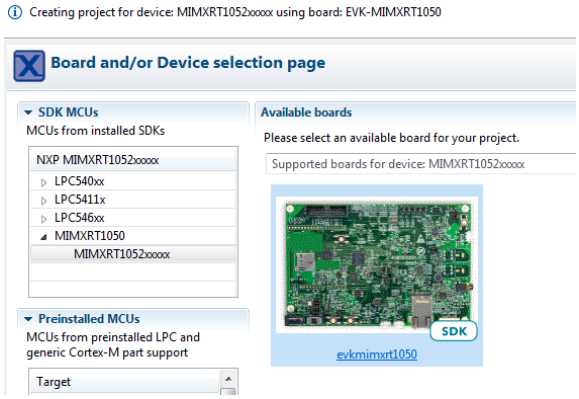
8 New Project Creation

The MCUXpresso IDE New Project wizard defaults to creating projects to execute in place (XIP) from the board HyperFlash using the SRAM_OC for data.

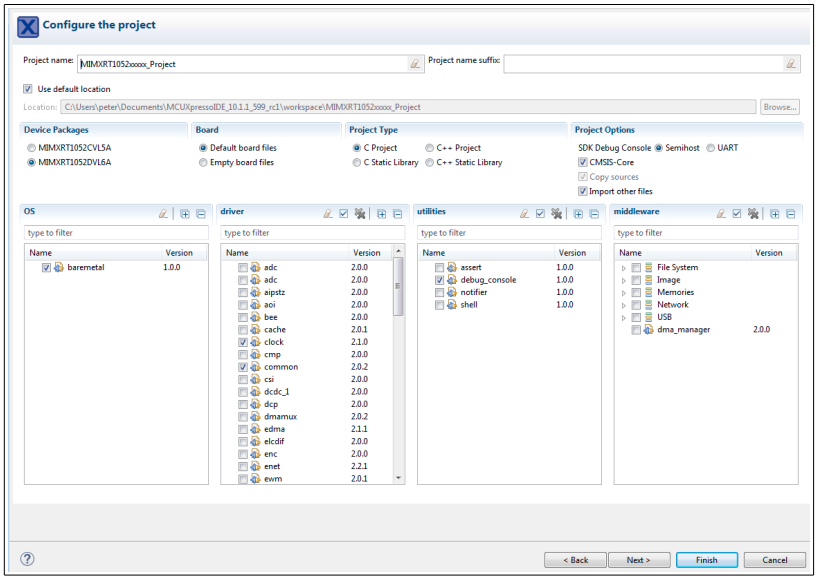
1 – To create a New Project, Click 'New Project' to launch the New Project Wizard:



2 - Ensure the EVK board is selected (otherwise board features including flash memory will not be available):

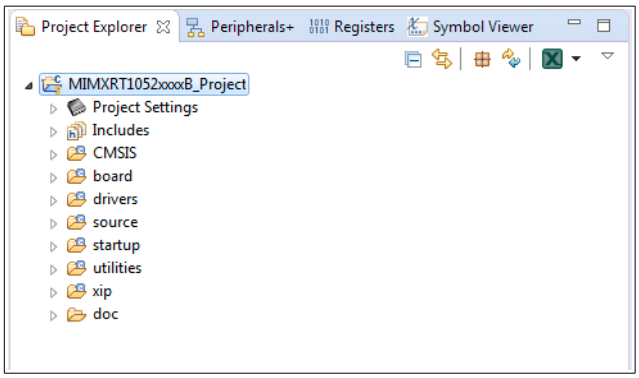


3 - Click Next (accepting all the default options):



4 - Then click Finish.

A new project (as below will be created).



This is a 'Hello World' project/application that will execute (XIP) from the Hyperflash memory using the first RAM region (SRAM_OC) for stack and global data. This RAM region is used because the SRAM_OC is marked by the SDK as the first RAM region for new projects (this selected RAM can of course be changed).

Note: Previous SDKs selected the SDRAM as the first RAM region. The SDRAM memory if used will be initialised by the BootROM (using the data from the XIP boot header) before being used by the application.

New Project issues

SDKs prior to version 2.4 had a number of issues impacting new project creation. These have been corrected in SDK 2.4.x. The information below (in grey) is supplied for reference.

For a project configured to XIP from hyper flash, a define **XIP_EXTERNAL_FLASH** should also be created. This define is used to select some clock setup for the flash and change its MPU cacheable properties. Currently, the SDK does not specify this define for new projects.

Without this symbol, the projects performance will be reduced.

Also, for projects using the SDRAM, a define **SDRAM_MPU_INIT** should also be set since this is used to determine the MPU properties for the SDRAM region. Currently, the SDK does not specify this define for new projects.

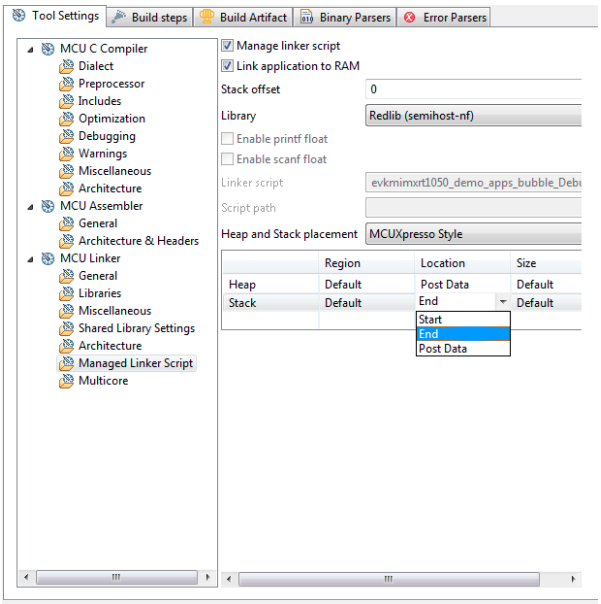
Without this symbol, the projects performance will be reduced.

Note: A new project define can be added in a number of ways, for example:

1. *Select the project in the Project Explorer.*
2. *From the QuickStart Panel -> Quick Settings -> Defined Symbols*
3. *Click +, enter the new symbol*
4. *Click OK, OK*

By default, the stack will be placed at the end of the first RAM region, therefore in our example project the stack will grow down from 0x82000000. However, the memory properties (as set in the MPU) for the last 2MB of the SDRAM are not optimal for stack operations. This problem can be corrected in a number of ways for example:

1. Edit the project properties and relocate the stack from the End of the memory region, to instead follow after the project's data – Post Data (as shown below).



2. Alternatively, edit the MPU description of the SDRAM region within the file board.c by simply deleting the Region 8 settings (in blue).

```
#if defined(SDRAM_MPU_INIT)
    /* Region 7 setting */
    MPU->RBAR = ARM_MPU_RBAR(7, 0x80000000U);
    MPU->RASR = ARM_MPU_RASR(0, ARM_MPU_AP_FULL, 0, 0, 1, 1, 0,
ARM_MPU_REGION_SIZE_32MB);

    /* Region 8 setting */
    MPU->RBAR = ARM_MPU_RBAR(8, 0x81E00000U);
    MPU->RASR = ARM_MPU_RASR(0, ARM_MPU_AP_FULL, 1, 0, 0, 0, 0,
ARM_MPU_REGION_SIZE_2MB);
#endif
```

9 XIP How and Why

Traditionally a standard Cortex M application image is programmed into an internal flash memory (of an MCU), this image is automatically executed once the MCU is reset (a bootable flash). Although essentially hidden from the user; when an MCU is reset, the first code to run is (usually) an internal BootROM, which is responsible for internal hardware setup and passing control to the users application in Flash. From the perspective of the user however, it appears as though their application is run immediately on reset.

In the case of the RT1050, all flash memory is external to the MCU and therefore unknown to the BootROM. For the BootROM to boot an image from this flash, some additional information must be supplied to allow flash initialisation and optimal configuration etc. The BootROM specification expects this configuration data to be located in an 8KB header at the start of the users image (application). An XIP image supplies this information in an 8KB header at the start image itself. Once programmed into flash, this information can be read by the BootROM using basic subset of flash operations.

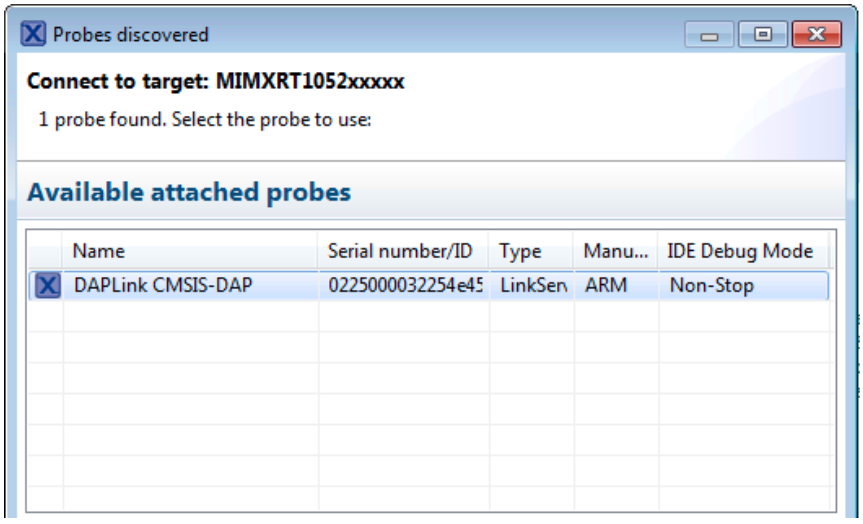
When the New Project Wizard is used and a board (evk(b)imxrt1050) is selected, board components will automatically be pre-selected - including an xip driver. This driver component will 'pull in' the required header files into a project folder (xip). The files within this folder work in conjunction with our Managed Linker Script mechanism to create and locate an appropriate header for this target flash device.

Note: this image header will only be created if the image is linked to the start of Hyperflash at 0x60000000 (the New Project default).

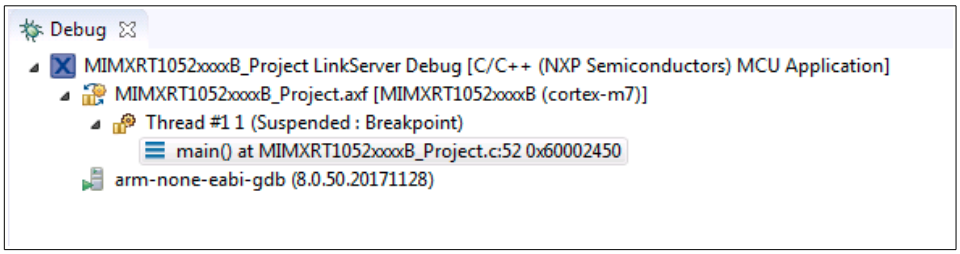
10 Project Debug

To debug this new project, check the section at the start of this document and ensure the board is correctly configured and powered. Then simply select the Project and from the Quickstart panel, click Debug.

A probe discovery operation will be performed, which should locate the on board DAPLink debug probe. Select this and click OK.



The project will build and a debug operation will commence. If all goes well, you should see the following Debug stack and the application halted on main().



The debug operations are logged (within the Console → Debug Messages) and will look as below:

```
MCUXpresso IDE RedlinkMulti Driver v10.2 (Jun 28 2018 13:51:56 -
crt_emu_cm_redlink build 554)
Found chip XML file in
C:/Users/peter/Documents/MCUXpressoIDE_10.2.1_792_alpha/workspace1/MIMXRT10
52xxxxB_Project/Debug\MIMXRT1052xxxxB.xml
Reconnected to existing link server
Connecting to probe 1 core 0:0 (using server started externally) gave 'OK'
Probe Firmware: DAPLink CMSIS-DAP (ARM)
Serial Number:  0227000041114e45004b3003b60f003aa6e1000097969900
VID:PID:  0D28:0204
USB Path: \\?\hid#vid_0d28&pid_0204&mi_03#8&1745eda8&0&0000#{4d1e55b2-f16f-
11cf-88cb-001111000030}
Using memory from core 0:0 after searching for a good core
debug interface type       = Cortex-M7 (DAP DP ID 0BD11477) over SWD TAP 0
processor type             = Cortex-M7 (CPU ID 00000C27) on DAP AP 0
number of h/w breakpoints = 8
number of flash patches    = 0
number of h/w watchpoints = 4
Probe(0): Connected&Reset. DpID: 0BD11477. CpuID: 00000C27. Info: <None>
Debug protocol: SWD. RTCK: Disabled. Vector catch: Disabled.
Content of CoreSight Debug ROM(s):
RBASE E00FD000: CID B105100D PID 000008E88C ROM dev (type 0x1)
ROM 1 E00FE000: CID B105100D PID 04000BB4C8 ROM dev (type 0x1)
ROM 2 E00FF000: CID B105100D PID 04000BB4C7 ROM dev (type 0x1)
ROM 3 E000E000: CID B105E00D PID 04000BB00C ChipIP dev SCS (type 0x0)
ROM 3 E0001000: CID B105E00D PID 04000BB002 ChipIP dev DWT (type 0x0)
ROM 3 E0002000: CID B105E00D PID 04000BB00E ChipIP dev (type 0x0)

ROM 3 E0000000: CID B105E00D PID 04000BB001 ChipIP dev ITM (type 0x0)
ROM 2 E0041000: CID B105900D PID 04001BB975 ARCH 23B:4A13r0 CoreSight dev
type 0x13 Trace Source - core
ROM 2 E0042000: CID B105900D PID 04004BB906 CoreSight dev type 0x14 Debug
Control - Trigger, e.g. ECT
ROM 1 E0040000: CID B105900D PID 04000BB9A9 CoreSight dev type 0x11 Trace
Sink - TPIU
ROM 1 E0043000: CID B105F00D PID 04001BB101 System dev (type 0x0)
Inspected v.2 External Flash Device on SPI MIMXRT1050-EVK_S26KS512S.cfx
Image 'MIMXRT1050-EVK_S26KS512S Jul 17 2018 18:22:12'
Non-standard DAP stride detected - 1024 bytes
NXP: MIMXRT1052xxxxB
Connected: was_reset=true. was_stopped=false
Awaiting telnet connection to port 3333 ...
GDB nonstop mode enabled
Opening flash driver MIMXRT1050-EVK_S26KS512S.cfx
Sending VECTRESET to run flash driver
Writing 25020 bytes to address 0x60000000 in Flash
Erased/Wrote page  0-0 with 25020 bytes in 1387msec
Closing flash driver MIMXRT1050-EVK_S26KS512S.cfx
Flash Write Done
Flash Program Summary: 25020 bytes in 1.39 seconds (17.62 KB/sec)
Starting execution using system reset and halt target
```

```
Note - system reset leaves VTOR at 0x200000 (not 0x60000000 which a booted
image might assume)
Stopped: Breakpoint #1
```

11 SDK Examples

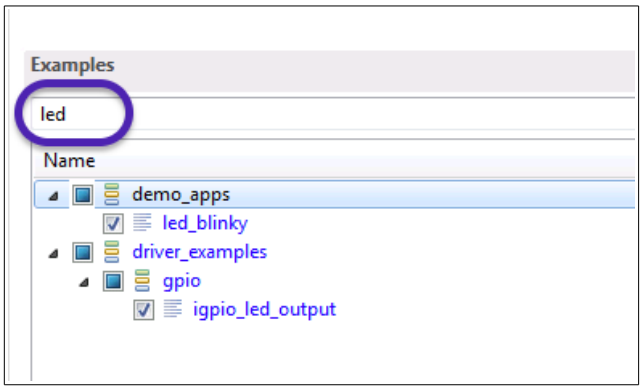
The *EVKB-IMXRT1050 SDK version 2.4.x* contains many examples, the majority of which target execution (XIP) from hyperflash.

Note: Examples from earlier SDKs typically targeted RAM regions and required modification to executed from hyperflash.

Importing an example

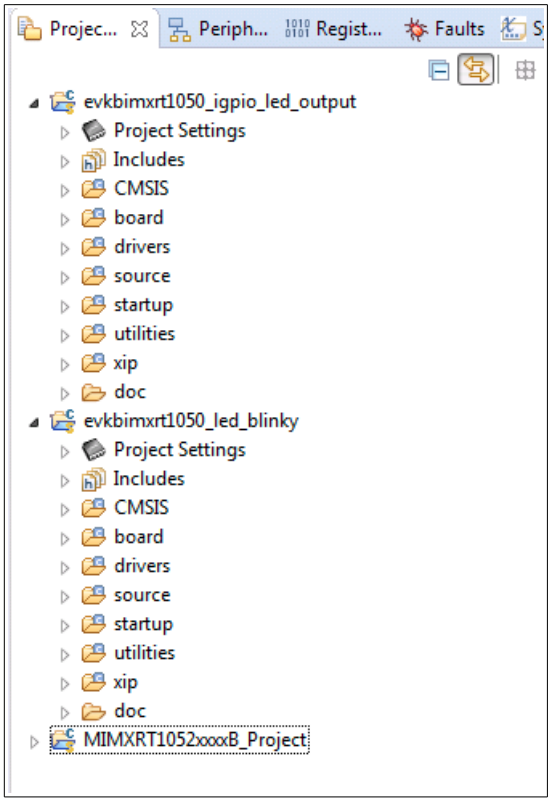
From the QuickStart panel select the Import SDK example wizard. Ensure the Board is selected as in the previous wizard and click Next.

Use the Filter to quickly locate the required example. For example: type 'led' as shown below:



Select the required projects and Click Finish.

The Project Explorer will look as below:



Note: the two projects demonstrate a simple delay based blinky and an interrupt based version. Both import the required XIP header and executed from Flash.

This project(s) can be debugged directly and you should see the board's LED flash. If the board is power cycled, you should see the LED flash program re-run from Flash.

12 Resets

When is a reset not a reset ...

The standard way a project is debugged is (after flash programming) for the MCU to be reset (by the debug probe) and user debug control made via an automatic breakpoint set on main(). This scheme though will only work if the application being debugged can be launched via the BootROM. In our case above, that would be an XIP image in Hyperflash with a correct header to describe the world and initialise the hyper flash.

However, it can also be useful to develop and debug applications running directly in a RAM region. For this to work the user must still gain control of the executing code i.e. via a breakpoint on main() again. However, a real reset cannot be used since this will run the BootROM and this will control the boot process and not lead us to our RAM location ...

Instead for RAM projects, the debugger will issue a virtual reset using the type SOFT. A SOFT reset type, simulates some parts of a real reset including setting up the PC, SP, PSR etc. Since this is not a real reset, the MCU hardware can inherit some setup from its (pre reset)

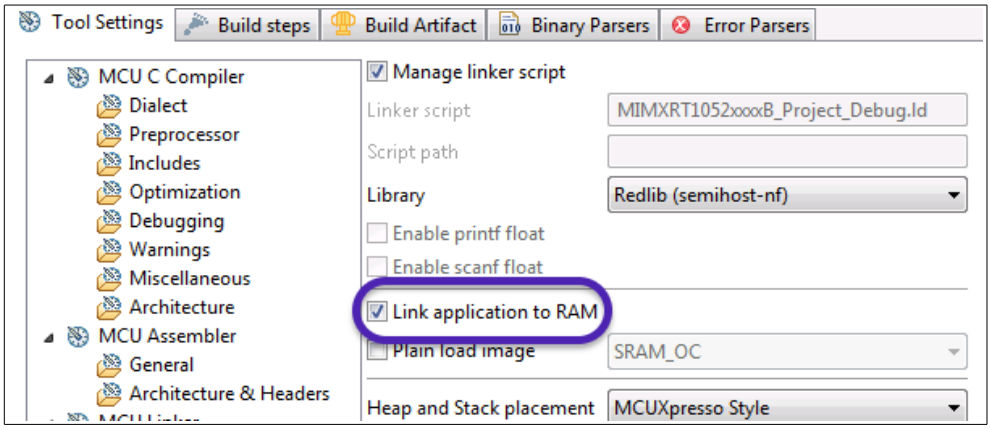
world, for example multiplexing, RAM and/or FLASH configurations. Note: This may be beneficial, but may also cause problems or confusion if not well understood.

SDK projects that target RAM use this SOFT reset mechanism.

Note: A project running from RAM may not restart successfully since Global data may only have the intended initial values on the first execution. This is a consequence of the mechanism and not a fault as such. The expected result can be achieved by using the QuickStart 'Terminate, Build and Debug' feature. All information will of course be lost if the target board is powercycled.

13 Building and Debugging Projects for RAM

Some example projects (for the iMX RT1050) are configured to link to RAM. This can been achieved using an MCUXpresso IDE feature that ensures any defined Flash region is ignored by the Linker resulting in an image being linked to the first defined RAM region.



Note: Some older SDK examples that target RAM execution target the DTCM RAM region at 0x20000000 using the SOFT reset type. They also make use of the above 'Link to RAM' feature while not actually including the definition of any Flash region - this feature is only intended for the case where flash is also present and is redundant if no flash is defined and may lead to user confusion (see the section below for more information).

Using the SDRAM region

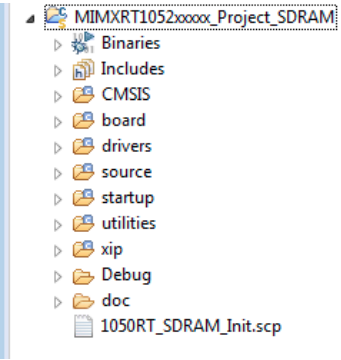
The TCM and OC memories should always be present when the part is powered on. However, the largest RAM on this board is the 32MB SDRAM. As mentioned above, this RAM is configured by the XIP header code for Flash projects - but for RAM projects to use this RAM area an alternative initialisation scheme is needed.

Note: If you wish to target the SDRAM region, this must be the first RAM region defined in the projects memory configuration. If a flash region is also defined, the Link application to RAM option must also be set.

LinkServer debug operations can run script files at the debug connect stage and also at the debug reset stage. In this situation, a connect script can be used to enable the SDRAM -

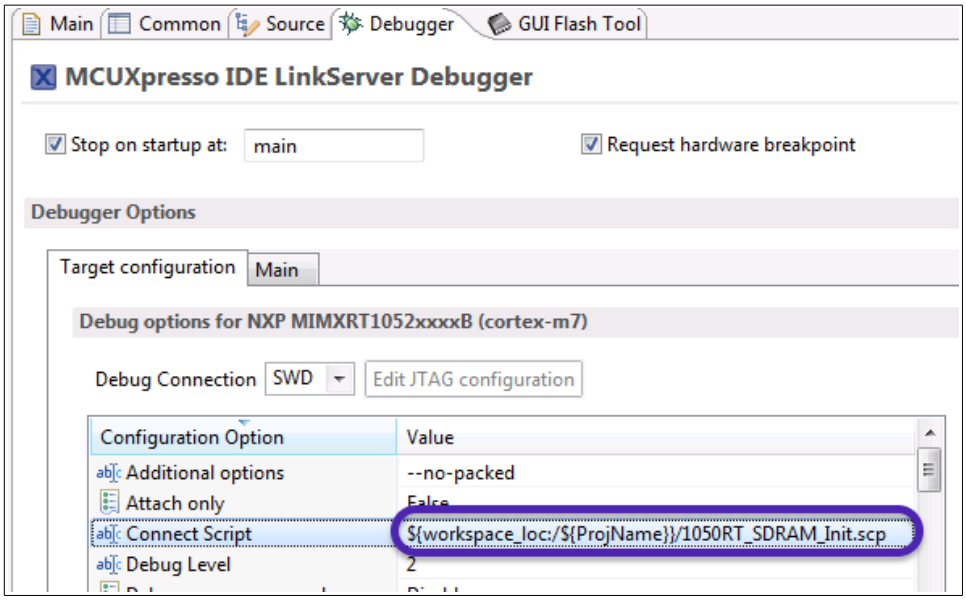
therefore making this available for an image to download into the RAM (at 0x80000000). Such a script 1050RT_SDRAM_Init.scp.

The appropriate script can be dropped directly into the project folder (as below) or copied into the product itself at *<install dir>/ide/bin/Scripts.*



To use the Script, edit the existing LinkServer Launch configuration (double click) and click inside the Connect Script Value field to browse for the Script. Select the Script and click OK.

Note: to create a new launch configuration either perform a debug operation via a LinkServer/CMSISDAP probe or right click on the project and select Launch Configurations -> Create new... -> MCUXpresso IDE LinkServer ...



When run, the script will cause the following output in the connections Debug log.

```
============= SCRIPT: RT1050_SDRAM_Init.scp =============
Setup SDRAM
Clock Init Done
SDRAM Init Done
============= END SCRIPT ================================
```

and of course SDRAM memory will be available for image download.

New and example (board) projects will contain a function *BOARD_ConfigMPU* within the file board.c . This function performs the MPU configuration for the various memory regions including cache setup and the exact behaviour of this function is controlled by a number of defined symbols.

It is strongly recommended that (if used) this function is examined and understood to ensure the memory system is configured as desired.

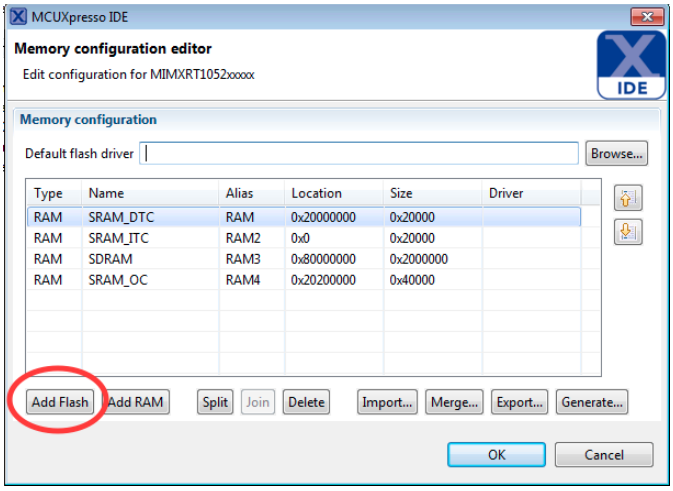
14 SDK Examples: Converting a RAM project to XIP from Flash

Project built to run from RAM will of course be lost if power is removed however they can be converted to run from Flash (XIP) by following the procedure outlined below.

There are three (or four!) steps to convert an image for XIP from Flash.

1 - Change the projects memory configuration to include the HyperFlash region and add a flash driver.

Right click on the project and select Properties -> C/C++ Build -> MCU settings -> Edit
Click Add Flash

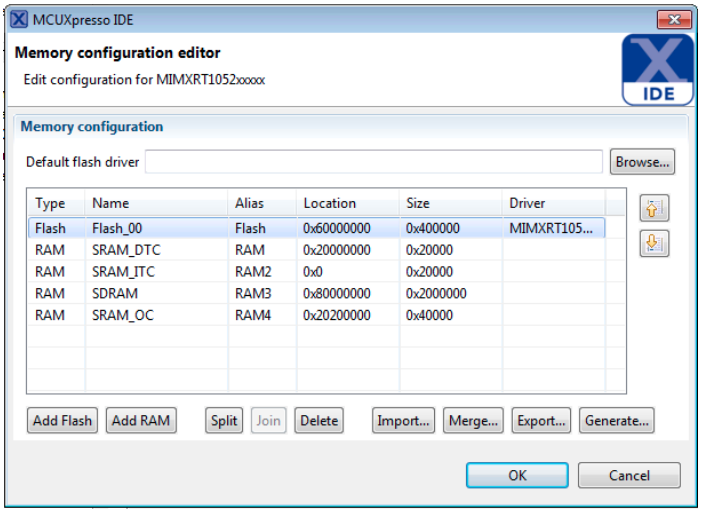


The hyper flash on this board is located at 0x60000000 and is 64MB in size.

Enter the new Location address at 0x60000000, and the Size as 0x400000 (or 64M), then click in the Driver field to select the flash driver 'MIMXRT1050-EVK_S26KS512S.cfx. **Be sure to click outside of the edited field (i.e. inside a blank field) to ensure the edit completes.**

Your memory configuration will look like below:

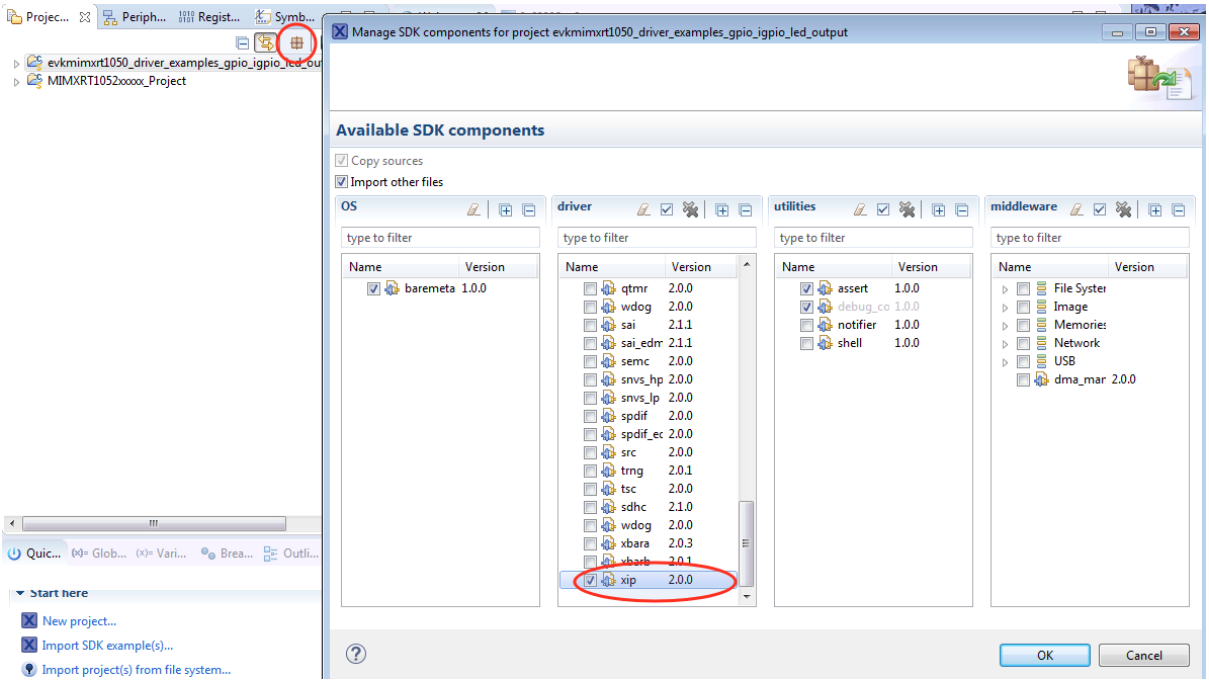
Note: the first RAM (SRAM Data TCM in this case) will automatically be used for the project's stack and global memory usage. The order of the memory regions can be changed using the right hand (up/down) buttons if required.



Click OK.

2 - Add the XIP files.

To add XIP files to a project, select the Project and then click the Manage SDK Components as shown below:



3 - Add the symbol definition(s) as discussed above.

Right click on the project and select Properties -> C/C++ Build -> Settings -> Preprocessor -> click + and add the new define – **XIP_EXTERNAL_FLASH=1**

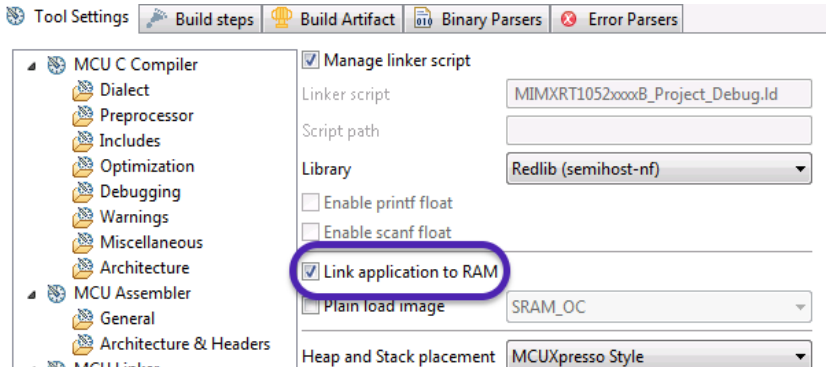
4 – Restore Linkage to Flash (if required).

For projects built to execute from Flash (the historically normal case) - there is a quick setting that can be used to force a project to instead link against the first RAM bank. The

SDK is causing this feature to be set even when no Flash is configured. Hence when the project is rebuilt, it will link still to RAM even though we have just defined a Flash region.

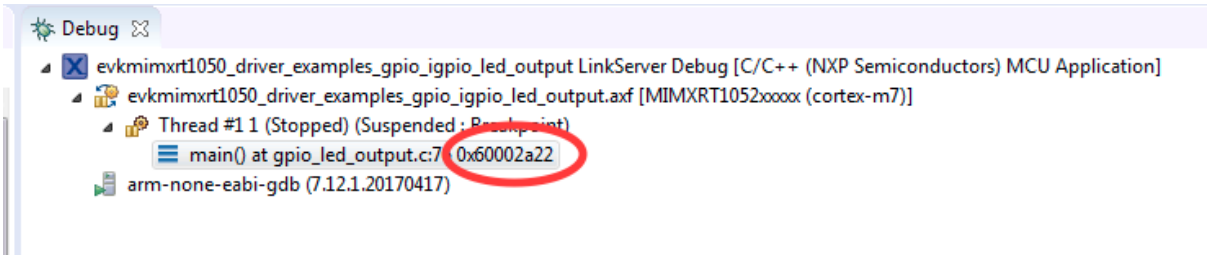
To restore linkage to Flash:

Right click on the project and select Properties -> C/C++ Build -> Settings -> Managed Linker Scripts -> Uncheck - Link application to RAM



Now, debug this application as before:

You should see the PC stopped at a Hyperflash address i.e. in the 0x60000000 range as below:



Note: as discussed above, projects targeting RAM will use a SOFT reset type. This reset type is assumed by the IDE to be the default for RAM projects however, some examples may explicitly set this reset type within the launch configuration. Therefore, if a RAM based project already has a launch configuration this SOFT reset type may remain after the conversion to XIP. The simplest way to fix this problem is to delete any existing launch configuration from the project and allow a new one to be automatically re-created. Alternatively, it can be edited and the SOFT option removed leaving no entry for reset type.

Booting the Application

Switch off power to the board (or unplug the OpenSDA debug connection). Remember to ensure the DIP switch (SW7) is set boot from Hyperflash, then reapply power. The board should boot the image.

Note: examples that use semihosting for printf, can still run without debug support if a hardfault handler 'semihost_hardfault.c' is included within the image. This handler is usually included by default.

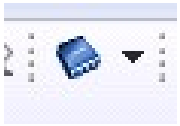
15 Troubleshooting

Erasing the Hyperflash

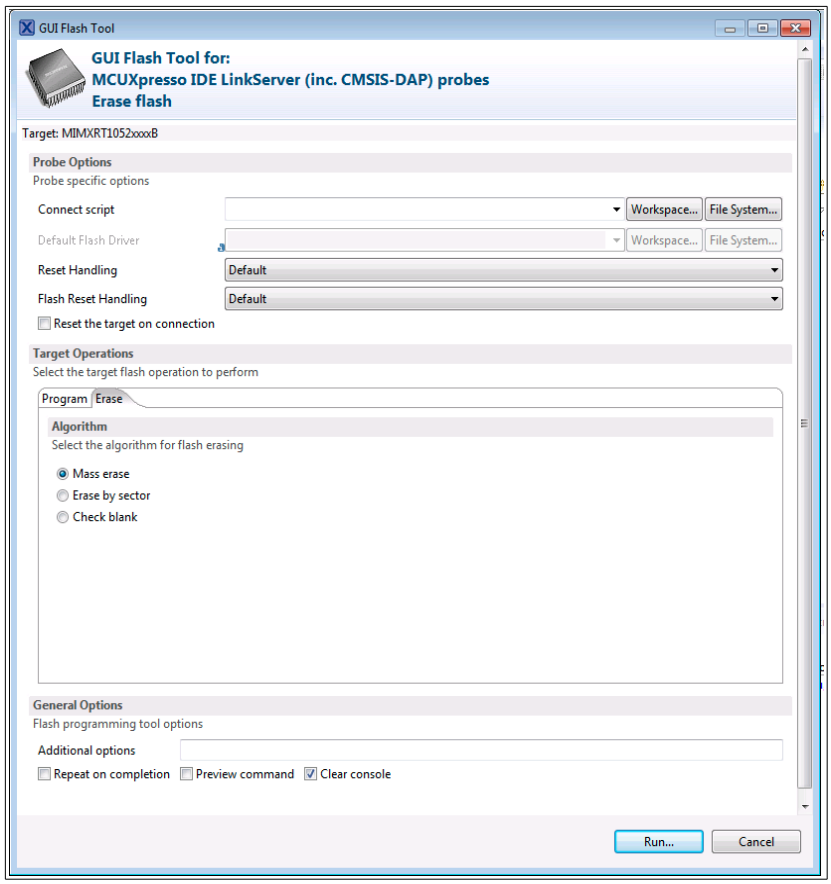
If for any reason a 'bad' application is programmed into flash and the BootROM boots this image, the resulting executed code may affect the part in such a way that prevents new debug operations succeeding. Should this occur, the following operation should recover the situation.

1. Power off the board.
2. Change the DIP switch (SW7) to prevent booting from hyperFlash e.g. set 1- on.
3. As a precaution, kill any active debug components
   ➔ to do this, click the icon:

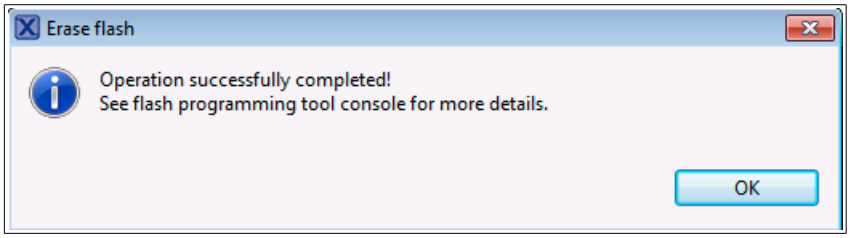
4. Use the GUI Flash Tool to Erase the data in HyperFlash.
   ➔ to do this, select a project that is configured for XIP from flash (e.g. a New project as described above) and click the GUI Flash Tool icon (the chip):



5. Select the probe as for a normal debug operation, then ensure the 'Erase flash memory' tab is selected. Click the 'Mass erase' radio button and then click OK.



6. This will cause a mass erase of the hyper flash, leading to the following dialogue.



Remember to restore the DIP switch (SW7) to set boot from Hyperflash and also power cycle the board. Next time the board boots, the BootROM will identify the Flash as erased and avoid running any flash image.

Note: The time taken to erase this flash is proportional to the current contents and may take some minutes to complete.

Obtaining debug control via a debug Connect Script

It has been observed that debug control may be impacted by certain applications running in Flash. This problem can manifest in being unable to make a debug connection – if this occurs, even a mass erase may be impossible.

As discussed earlier in this document, LinkServer debug operations can run script files at the debug connect stage, a script to work around connection problems called RT*1050_Debug_Connect.scp* is supplied with this document.

This script can be dropped directly into the project folder or copied into the product itself at *<install dir>/ide/bin/Scripts* and then referenced in a debug launch configuration.

Using external Debug Probes

The onboard DAPLink OpenSDA debug interface does not deliver particularly high debug performance. The standard 20way 'JTAG' header may be used with external debug probes such as the LPC-Link2 for faster debug operation.

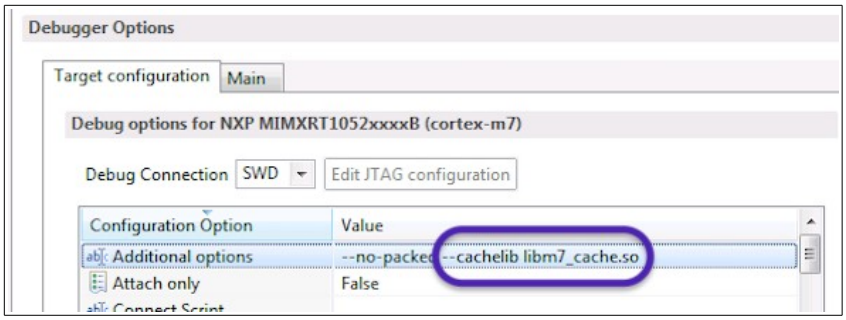
If the LPC-Link2 is used it is recommended that the board be both powered externally and the Link JP2 should be fitted to the LPC-Link2 probe.

Please note: It has been observed that debug via these external debug probes may be less reliable with certain images. Such debug problems will manifest as *wire ack* faults. This problem may be improved if the OpenSDA debug interface is disabled, please see Overview of Board features … point (6) for more information.

If these are seen it is recommended that the OpenSDA debug connection is used.

Debug performance and the Data Cache

When debugging images that make use of SDRAM or OC_RAM for storage of variable data (globals, stack, heap etc.) then the following option should be set within the LinkServer debug launch configuration as shown below:



This module ensures that debug cache coherence is maintained, and correct debug operations may fail if this module is not specified. However there will be a debug performance penalty when this module is used.

Note: this module is not required if the SDRAM (or OC_RAM) only contains constant or uncached data.